

Lecture 1

CIS 5000: SOFTWARE FOUNDATIONS

Steve Zdancewic

Fall 2022

COVID

- Follow the Penn Covid Guidelines
 - <https://coronavirus.upenn.edu/>
- For now, we REQUIRE you to wear properly-fitted masks for lectures, office hours, etc.
- If you contract COVID
 - follow the university procedures for isolation, quarantine
 - let us know about your status
 - course lectures are recorded, material is online
 - accommodations for HW, exams, participation, etc.

Our primary concern is the health and wellbeing of *all* students, and facilitating their success in this class.

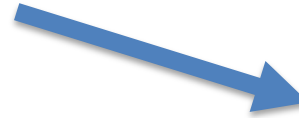
SOFTWARE FOUNDATIONS



How do we build software?

that works
(and be convinced
that it does)

Critical Software



Individual programs Programming languages

- Operating systems
- Network stacks
- Crypto
- Medical devices
- Flight control systems
- Power plants
- Home security
- Blockchain
- ...

- Compilers
- Static type system
- Data abstraction and modularity features
- Security controls

Logic

+ Reasoning about
individual programs

+ Reasoning about
whole programming
languages

SOFTWARE FOUNDATIONS



LOGICAL FOUNDATIONS



Q: How do we know something is true?

A: We prove it.

Q: How do we know that we have a *proof*?

A: We need to know what it means for something to be a proof.

First cut: A proof is a “logical” sequence of arguments, starting from some initial assumptions.

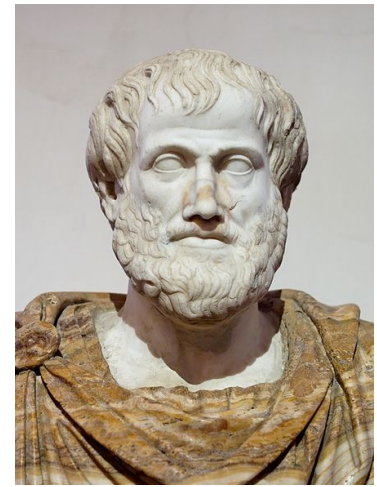
Q: How do we agree on what is a *valid* sequence of arguments? Can any sequence be a proof? E.g.

All humans are mortal

All Greeks are human

Therefore, I am a Greek!

A: No, no, no! We need to think harder about valid ways of reasoning...



Aristotle
384 – 322 BC



Euclid
~300 BC

First we need a *language*...

- **Gottlob Frege**: a German mathematician who started in geometry but became interested in logic and foundations of arithmetic.
- 1879 Published "*Begriffsschrift, eine der arithmetischen nachgebildete Formelsprache des reinen Denkens*" (Concept-Script: A Formal Language for Pure Thought Modeled on that of Arithmetic)
 - First rigorous treatment of functions and quantified variables
 - $\vdash A, \neg A, \forall x.F(x)$
 - First notation able to express arbitrarily complicated logical statements



Gottlob Frege
1848-1925



Formalization of Arithmetic

Frege made great progress, introducing:

- 1884: *Die Grundlagen der Arithmetik* (The Foundations of Arithmetic)
- 1893: *Grundgesetze der Arithmetik* (Basic Laws of Arithmetic, Vol. 1)
- 1903: *Grundgesetze der Arithmetik* (Basic Laws of Arithmetic, Vol. 2)

Frege's goals:

- isolate logical principles of inference
- derive laws of arithmetic from first principles
- set mathematics on a solid foundation of logic

The plot thickens...

Just as Volume 2 was going to print in 1903,
Frege received a letter...

Addendum to Frege's 1903 Book

“Hardly anything more unfortunate can befall a scientific writer than to have one of the foundations of his edifice shaken after the work is finished. This was the position I was placed in by a letter of Mr. Bertrand Russell, just when the printing of this volume was nearing its completion.”

– Frege, 1903

Bertrand Russell

- *Russell's paradox:*

1. Set comprehension notation:

$\{ x \mid P(x) \}$ “The set of x such that $P(x)$ ”

2. Let X be the set (of sets) $\{ Y \mid Y \notin Y \}$.

3. Ask the logical question:

Does $X \in X$ hold?

4. **Paradox!** If $X \in X$ then $X \notin X$.
If $X \notin X$ then $X \in X$.



Bertrand Russell
1872 - 1970

- Frege's language could derive Russell's paradox \Rightarrow it was *inconsistent*.
- Frege's logical system could derive anything. (Oops!)

David Hilbert

German recognized as one of the most influential mathematicians ever.

- studied algebra, axiomatization of geometry, physics,...

1900: published his "23 Problems"

- Problem #2: Prove that the axioms of arithmetic are *consistent*

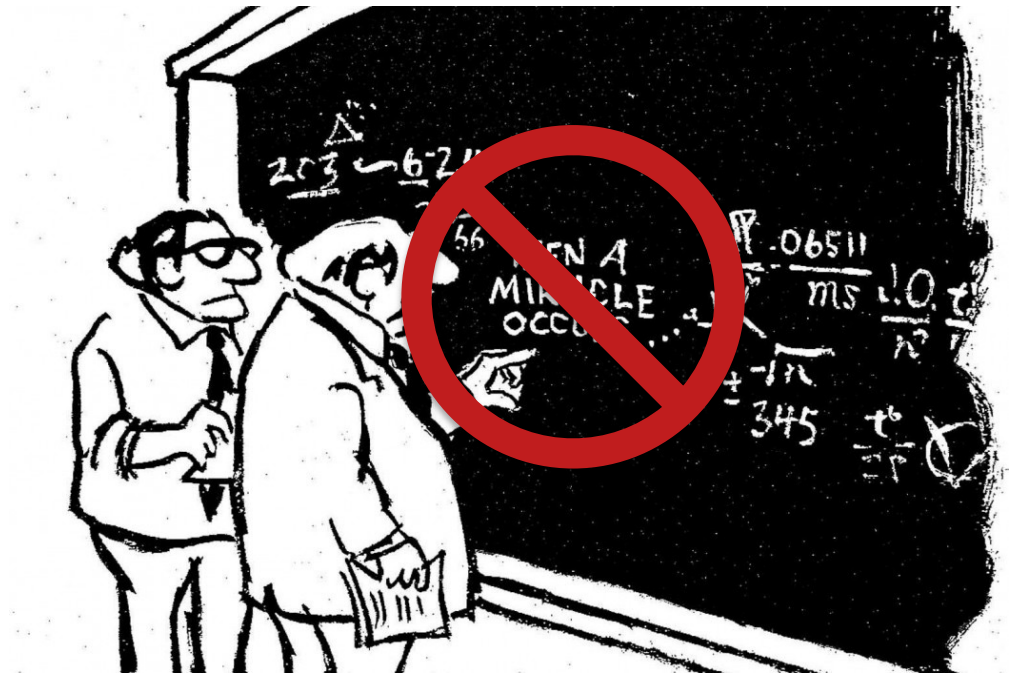


David Hilbert
1862 – 1943

1920's: Hilbert's Program

A plan to secure the foundations of mathematics:

- Develop a *formal system* of all mathematics.
 - Mathematical statements should be written in a precise formal language
 - Mathematical proofs should proceed by well-specified rules
- Prove *completeness*
 - i.e., that all true mathematical statements can be proved
- Prove *consistency*
 - i.e., that no contradictory conclusions can be proved
- Prove *decidability*
 - i.e., there should be an algorithm for determining whether a given statement has a proof



Aftermath of Frege and Russell

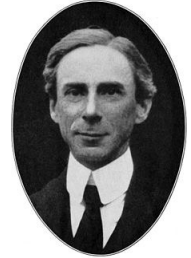
- Frege came up with a fix... but it made his logic trivial :-(
- **1908**: Russell fixed the inconsistency of Frege's logic by developing a *theory of types*.
- **1910, 1912, 1913**, (revised **1927**):
Principia Mathematica (Whitehead & Russell)
 - Goal: axioms and rules from which *all* mathematical truths could be derived.
 - It was a bit unwieldy...

"From this proposition it will follow,
when arithmetical addition has been defined,
that $1+1=2$."

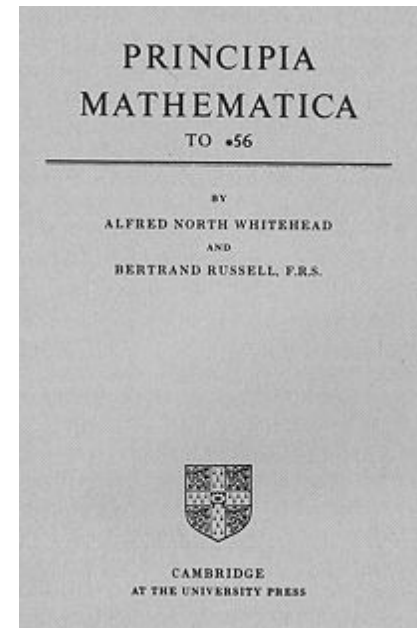
—Volume I, 1st edition, *page 379*



Whitehead



Russell



Nevertheless, things were going well,
following Russell & Whitehead, until...

Logic in the 1930s and 1940s

- 1931: Kurt Gödel's first and second *incompleteness theorems*.
 - Demonstrated that any consistent formal theory capable of expressing arithmetic cannot be complete.
 - Big idea: write down "This statement is not provable." as an arithmetic statement.
- 1936: Gentzen proves *consistency* of arithmetic.
- 1936: Church introduces the *λ -calculus*.
- 1936: Turing introduces Turing machines
 - Is there a decision procedure for arithmetic?
 - Answer: no, it's undecidable
 - The famous "halting problem"
 - n.b.: only in 1938 did Turing get his Ph.D.
- 1940: Church introduces the *simple theory of types*



Kurt Gödel
1906 - 1978



Gerhard Gentzen
1909 - 1945



Alonzo Church
1903 - 1995



Alan Turing
1912 - 1954

Fast Forward...

- Two logicians in 1958 (Haskell Curry) and 1969 (William Howard) observe a remarkable correspondence:



Haskell Curry
1900 – 1982



William Howard
1926 –

| | | |
|-------------|---|----------------|
| types | ~ | propositions |
| programs | ~ | proofs |
| computation | ~ | simplification |



N.G. de Bruijn
1918 - 2012

- 1967 – 1980's: N.G. de Bruijn runs Automath project
 - uses the Curry-Howard correspondence for computer-verified mathematics

- 1971: Jean-Yves Girard introduces System F
- 1972: Girard introduces F_ω
- 1972: Per Martin-Löf introduces intuitionistic type theory
- 1974: John Reynolds independently discovers System F

Basis for modern
type systems:
OCaml, Haskell,
Scala, Java, Rust,
Swift, ...

... to the Present

- 1984: Coquand and Huet first begin implementing a new theorem prover “Coq”
- 1985: Coquand introduces the **calculus of constructions**
 - combines features from intuitionistic type theory and $F\omega$
- 1989: Coquand and Paulin extend CoC to the **calculus of inductive constructions**
 - adds “inductive types” as a primitive
- 1992: Coq ported to Xavier Leroy’s OCaml
- 1990’s: up to Coq version 6.2
- 2000-2015: up to Coq version 8.4
- 2013: Coq receives ACM Software System Award
- 2022: Coq version 8.15.2 ← CIS 5000



Thierry Coquand
1961 –



Gérard Huet
1947 –

Too many contributors
to list here...

and, many other provers:

- Isabelle/HOL
- Agda
- LEAN
- ...

So much for foundations... what about the “software” part?

(LANGUAGE)
PROGRAMMING FOUNDATIONS




Building Reliable Software

- Suppose you work at (or run) a software company.
- Suppose, like Frege, you've sunk 30+ person-years into developing the "next big thing":
 - Boeing Dreamliner2 flight controller
 - Autonomous vehicle control software for Nissan
 - Gene therapy DNA tailoring algorithms
 - Super-efficient green-energy power grid controller
 - The next big blockchain
- Suppose, like Frege, your company has invested a lot of material resources that are also at stake.
- How do you avoid getting a letter like the one from Russell?

Or, worse yet, *not* getting the letter,
with disastrous consequences down the road?

Approaches to Software Reliability

- **Social**
 - Code reviews
 - Extreme/Pair programming
- **Methodological**
 - Design patterns
 - Test-driven development
 - Version control
 - Bug tracking
- **Technological**
 - “lint” tools, static analysis
 - Fuzzers, random testing
- **Mathematical**
 - Sound type systems
 - Formal verification



Less “formal”: Lightweight, inexpensive techniques (that may miss problems)

This isn’t a tradeoff... all of these methods should be used.

Even the most “formal” argument can still have holes:

- Did you prove the right thing?
- Do your assumptions match reality?
- Knuth: *“Beware of bugs in the above code; I have only proved it correct, not tried it.”*

More “formal”: eliminate *with certainty* as many problems as possible.

Can Formal Methods Scale?

Academia

- **Bedrock** – web; packet filters
- **CakeML** – SML compiler
- **CertiKOS** – certified OS kernel
- **CompCert** – C compiler
- **EasyCrypt** – crypto protocols
- **Kami** – RISC-V architecture
- **HS2Coq** – Library validation
- **SEL4** – OS microkernel
- **Vellvm** – LLVM IR
- **VST** – C software
- **Ynot** – DBMS, web services

Industry

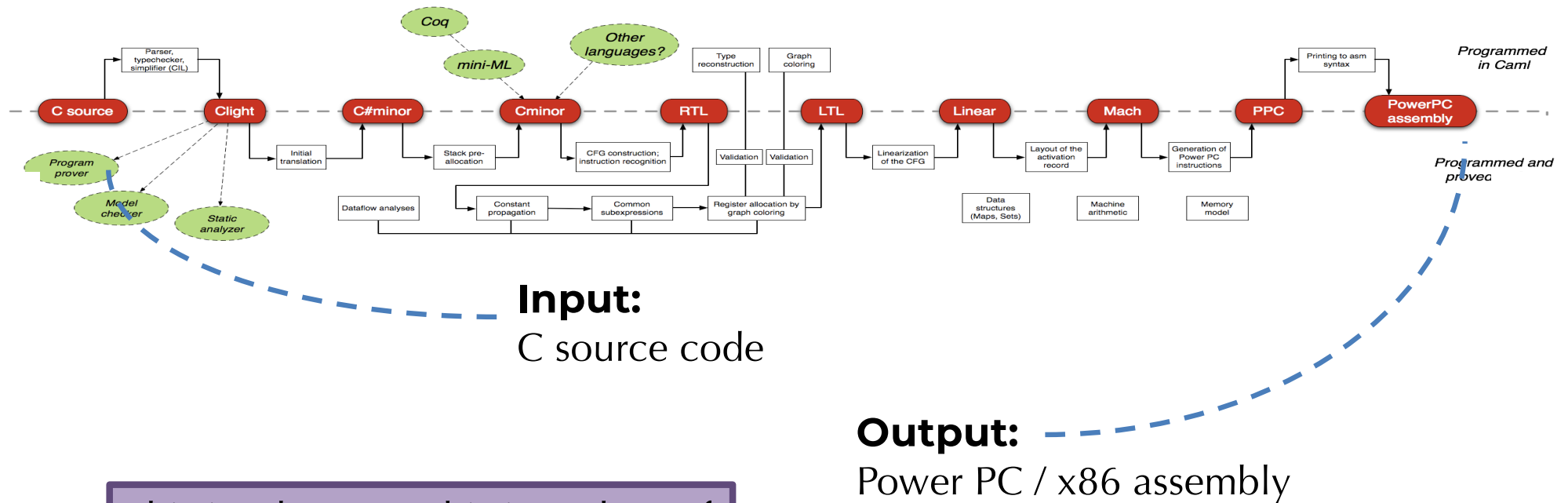


Flagship Example: CompCert

[Xavier Leroy, et al. INRIA, 2010 - present]

Optimizing C Compiler:

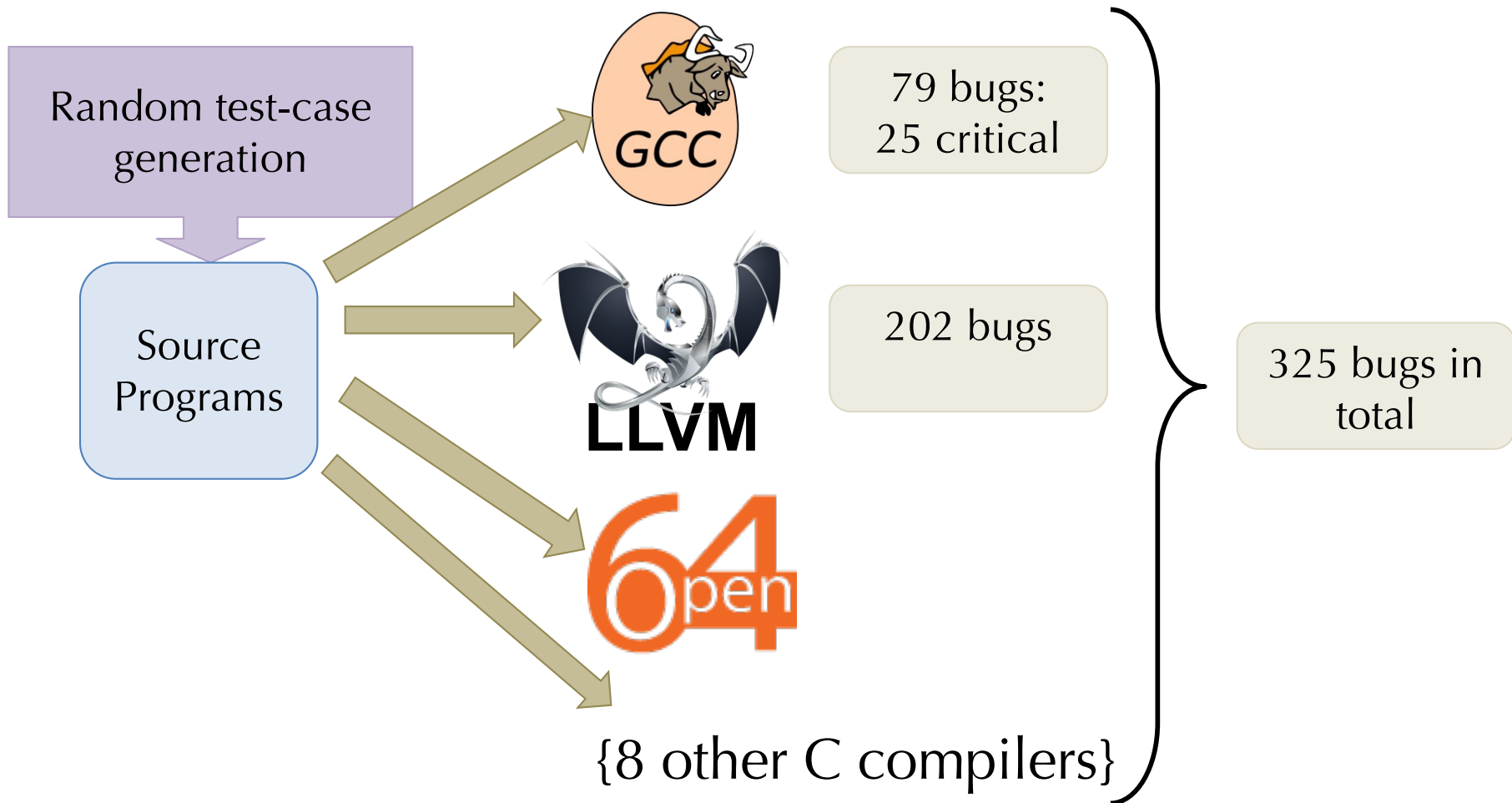
implemented and proved correct end-to-end with machine-checked proof in Coq



This is a large, sophisticated proof development, but ultimately it relies on the concepts introduced in CIS 5000.

Does it work?

Finding and Understanding Bugs in C Compilers [Yang et al. PLDI 2011]




CompCert

<10 bugs found only in the (at the time *unverified*) front-end component

Regehr's Group Concludes

The striking thing about our CompCert results is that the middle-end bugs we found in all other compilers are absent. As of early 2011, the under-development version of *CompCert is the only compiler we have tested for which Csmith cannot find wrong-code errors*. This is not for lack of trying: we have devoted about six CPU-years to the task. *The apparent unbreakability of CompCert supports a strong argument that developing compiler optimizations within a proof framework, where safety checks are explicit and machine-checked, has tangible benefits for compiler users.*

CIS 5000

- **Foundations**
 - Functional programming
 - Constructive logic
 - Logical foundations
 - Proof techniques for inductive definitions
- **Semantics**
 - Operational semantics
 - Modeling imperative “While” programs
 - Hoare logic for reasoning about program correctness
- **Type Systems**
 - Simply typed λ -calculus
 - Type safety
 - Subtyping
 - Dependently-typed programming
- **Coq interactive theorem prover**
 - turns doing proofs & logic into programming  fun!

COURSE MECHANICS

Administrivia

- **Instructor:** Steve Zdancewic
Office hours: See web page (currently Mondays 4-5)
Levine 511
- **TAs:** Jessica Shi, Adam Stein, and Joey Valez-Ginorio
- **E-mail:** cis5000@seas.upenn.edu (goes to all course staff)
- **Web site:** <http://www.seas.upenn.edu/~cis5000>
- **Infrastructure:**
 - Ed Discussion
 - Poll Everywhere (for in-class quizzes, not graded)
 - Gradescope (for homework submission)
 - Canvas (only to host zoom videos)

Resources

- **Course textbook:**

Software Foundations, volumes 1 and 2

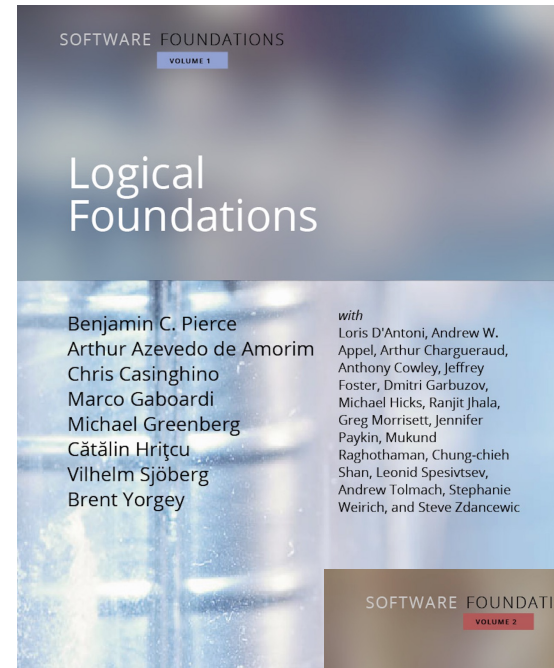
- Electronic edition tailor-made for this class

Use the version available from the cis5000 course web pages!!

(A new version of each chapter will generally go live just before class. :-)

- Additional resources:

- *Types and Programming Languages* (Pierce, 2002 MIT Press)
- *Interactive Theorem Proving and Program Development* (Bertot and Castéran, 2004 Springer)
- *Certified Programming with Dependent Types* (Chlipala, electronic edition)



How to CIS5000

Class participation is strongly encouraged

- Live lectures will be as interactive as possible!
- Ask lots of questions
- Focus on the class instead of multitasking

Lecture Recordings

- Every lecture will be recorded
- Should be available online a few hours later
- Feel free to use them (and the textbook) to supplement the material

Note: If you are not feeling well, or if you test positive for covid,
please do not attend lecture!

Class Participation

- We will be using Poll Everywhere, an online polling platform, for
 - in-class mini quizzes
 - real-time “polls” during lectures
 - (not graded!)
- For next time: *download the Poll Everywhere app for your smartphone.*

Course Policies

Prerequisites:

- Significant programming experience
- “Mathematical sophistication”
- Undergraduate functional programming or compilers class helpful

Grading:

- | | |
|-----------------|----------------------------|
| • 30% Homework | (~12 weekly assignments) |
| • 20% Midterm 1 | (in class, early October) |
| • 20% Midterm 2 | (in class, early November) |
| • 30% Final | (date TBA) |

“Regular” vs. “Advanced” Tracks

- **“Regular” track**
 - Usual mode of participation for Masters students and Undergraduates
- **“Advanced” track**
 - More and harder exercises
 - More challenging exams, emphasizing written proofs
 - Covers a superset of the “regular” material
- **Ph.D. Students *must* follow the advanced track.**
 - The advanced track final exam is the WPE I
 - Others wishing to take the advanced track (e.g., maybe a good idea if you plan to apply to PhD programs) should contact the course staff.
- “Regular” and “Advanced” tracks are **graded separately**
 - “Regular” track students are encouraged to try advanced-track problems.
 - A regular-track student's grades cannot be *harmed* by their performance on advanced-track material.
 - “Advanced” track problems are ***not*** bonus points and cannot replace regular track problems.
 - However: a consistent track record of doing advanced track material may earn you an improved over all letter grade

WPE-I Policy

- If you wish to take CIS5000 for WPE-I (Written Preliminary Exam, part I) credit toward a CIS PhD degree, you have two choices:
 - **Final exam only** option: WPE-I credit only (no need to be registered for the course). Passing score for WPE-I credit is determined by the CIS5000 instructors (Pierce, Weirich, Zdancewic). Historically, this has been around a B+ grade on the exam.
 - **Full course participation** option: Must be registered for the course. WPE-I credit awarded for a weighted average grade of B+ on homework and all three exams.
 - You can take the course P/F and also receive WPE-I credit (following the same criteria)

Homework

Structure

- Homework consists of Coq source files that you *complete individually*.
- Advanced track students must complete (or attempt) all non-optional exercises *including* those marked “advanced”.
 - Missing “advanced” exercises will count against your score.
- Regular track students must complete (or attempt) all non-optional exercises *except* those marked “advanced”.
 - Missing “advanced” exercises will *not* count against your score.
 - But you are welcome to try them!
- Subsequent homework assignments build on earlier ones
 - you should keep up with the course material

Note: the HW difficulty level ramps up significantly. If you struggle early, that is bad, but don't be deceived if you think it's easy.

Submission

- HW will be submitted to an autograder via Gradescope
- homework that does not compile will not be graded and will receive a 0

Late Homework

Late Policy

- HW cannot be submitted more than 48 hours late (unless you have unusual circumstances)
- HW submitted after the due date will accrue “late days,” one late day for each 24 hours, or fraction thereof, that the assignment is late.
- Any homework that is *not submitted* at all receives a score of 0 and accrues *2 late days*
- Late days are atomic, indivisible units, and partial late days are not permitted.

Accrued late days do not impact any individual homework score. Instead, they apply to the final weighted, letter grade of the class, as follows:

| | | |
|--------------------|--------------------------|---|
| 0 – 8 late days: | no grade penalty | |
| 9 – 16 late days: | one-third grade penalty | (A \Rightarrow A- B+ \Rightarrow B etc.) |
| 17 – 20 late days: | two-thirds grade penalty | (A \Rightarrow B+ B+ \Rightarrow B- etc.) |
| 21 – 24 late days: | full grade penalty | (A \Rightarrow B B+ \Rightarrow C+ etc.) |

Unusual Circumstances

If you have a medical, family, or other emergency

- Please ***contact the instructor as soon as possible***
(in advance of the due date or exam if you know ahead of time)
- If you do not contact the instructor in advance, it is up to their discretion whether you accrue late days due to the emergency

We will do our best to accommodate your unusual circumstances.

(The primary goal is for you to master the course material.)

Academic Integrity

Submitted material must be your own work.

Not OK

- Copying or otherwise looking at someone else's code
- Sharing your code in any way
(copy-paste, github, paper and pencil, ...)
- Using code from a previous semester

OK (and encouraged!)

- Discussions of concepts
- Discussion of debugging strategies
- Verbally sharing experience

Penn's code of academic integrity:

<http://www.upenn.edu/academicintegrity>

TODO (for you)

- Before next class:
 - Check out Ed Discussion (let us know if you are not already registered)
 - Try to log in to Gradescope
 - Download Poll Everywhere app on your phone
 - Install VSCode + Coq
 - Start reading: Preface and Basics
- HW1: Exercises in Basics.v
 - Due: Tuesday, September 6th at 10:00PM
 - Available from course web page
 - Complete all non-optional exercises
 - There are no “advanced” problems for this HW
 - Submit via Gradescope



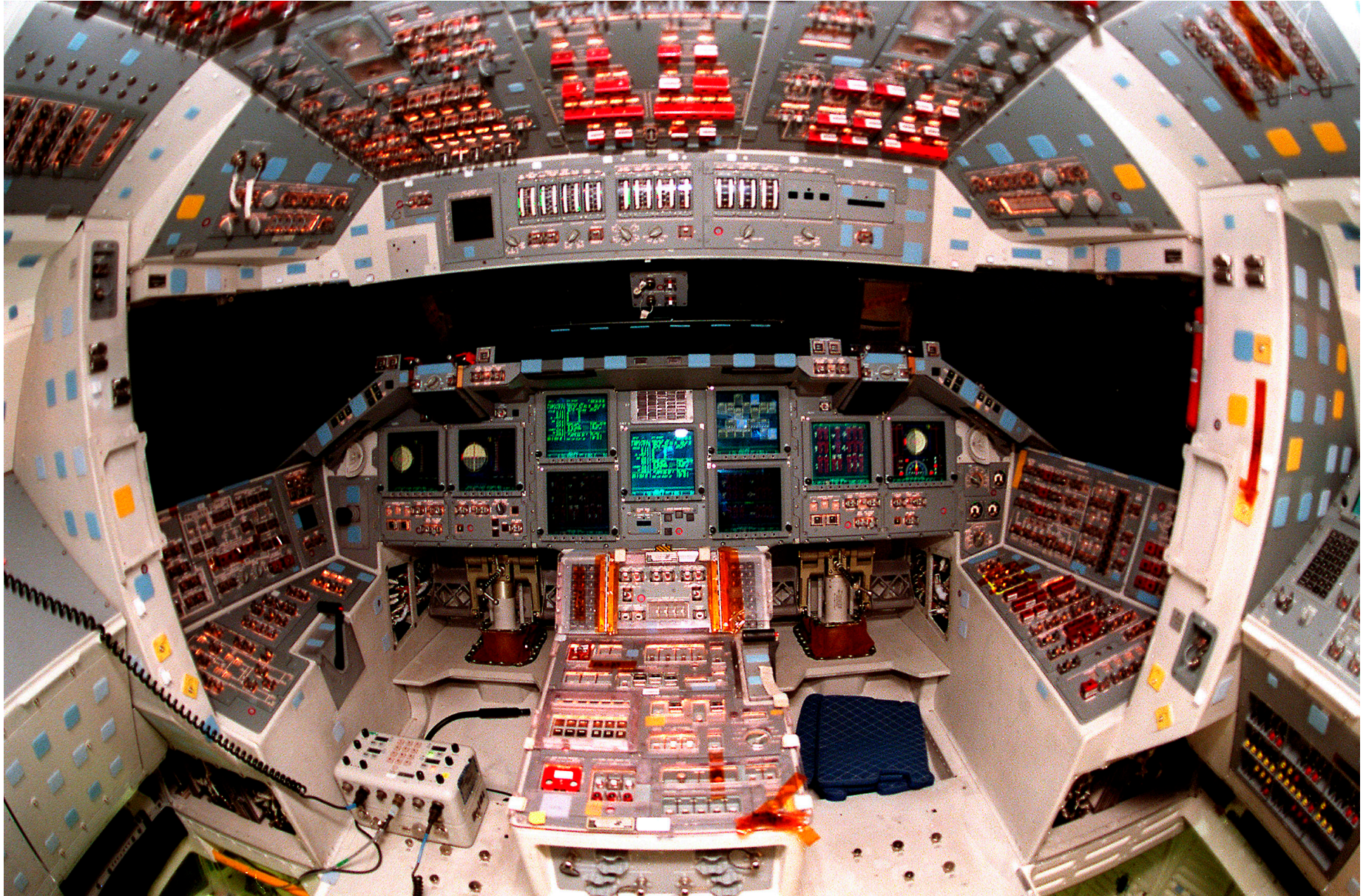
COQ

Coq in CIS 5000

- We'll use Coq version 8.15.2
 - We recommend using VSCode + Docker
 - Course web pages have installation instructions.
- See also the web pages at: coq.inria.fr
- Other available user interfaces
 - CoqIDE – a standalone GUI / editor
 - ProofGeneral – an Emacs-based editing environment



Coq's Full Capabilities



Subset Used in CIS 5000



To start.



By the end of the semester.

Getting acquainted with Coq...

BASICS.V